Institute for Science Networking Oldenburg GmbH

RDFEditor: User's Guide

Christian Thiemann*

February 07, 2005
Version 1.414213

Contents

1. The data model **2**
1.1. Preparing a data model for RDFEditor 2
1.2. Data model storage 2

2. Editor Configuration **3**
2.1. Data Model Options 3

3. The user interface elements **4**
3.1. Frames 5
3.2. Create pane 5
3.3. Resource list and hierarchical resource list 5
3.4. Resource edit pane 6
3.5. Container edit pane 8

4. User Management **8**

A. Example: Editor Configuration File **12**

*thiemann@isn-oldenburg.de

1. The data model

In the following it is assumed that the reader is familiar with *RDF (Resource Description Framework)*, *RDF Schema* and *OWL Lite (Web Ontology Language)*. An introduction into the concepts of RDF and RDF Schema can be found in the *RDF Primer* [1], the *OWL Reference* [3] introduces into the extensions of OWL Lite to RDF Schema.

1.1. Preparing a data model for RDFEditor

The RDFEditor requires metadata over the data model, which has to be provided as OWL. Actually, in the current version of RDFEditor only RDF Schema and the cardinality restrictions of OWL Lite are recognized.

The RDFEditor requires the metadata to be normative, that means that the data cannot contain instances of classes which are not declared in the metadata nor can it use undeclared properties. The RDFEditor also requires the cardinality restrictions to be obligatory.

The current version of the RDFEditor requires each resource to be an instance of exactly one class (i.e. that each resource has exactly one `rdf:type` property). It requires each property to have a domain and a range. This restrictions may be removed in future versions but it is not very likely since they enforce an object-orientated metadata design. For example, if you intend to declare a set of classes and each instance of any of the classes should be able to have a property called `name`, you have to declare an abstract class whose instances can be applied the `name` property and declare all your classes as subclasses of this abstract class. They will inherit the `name` property from the abstract class.

Section 2.1 on the following page explains how to design the data model in order to use all of RDFEditor's special features.

The current version of the RDFEditor has no real support for namespaces yet. It requires namespace prefixes in URIrefs and does not associate prefixes with URIs (thus two literally different prefixes are two different namespaces). This limitation will very likely be removed in future versions.

1.2. Data model storage

The only storage type supported by the current version is `mysql`. It is very likely that future versions of RDFEditor will support an `xml` storage type to export (and probably also import) RDF data to/from XML documents.

1.2.1. Using MySQL tables to store the data

When using MySQL tables as storage, the RDF data is serialized into RDF triples (subject, predicate, object) and each row in the MySQL table stores one triple. The RDFEditor expects MySQL tables storing RDF triples to be created as follows:

```
CREATE TABLE <table_name> (
    s BINARY VARCHAR(100) NOT NULL,
    p BINARY VARCHAR(100) NOT NULL,
    o BLOB NOT NULL,
    PRIMARY KEY(s,p,o(300))
)
```

If the data model is not case-sensitive, `TEXT` may be used instead of `BLOB` and the `BINARY` attributed may be omitted. The numbers may be adjusted to the needs of the data model. It is important to note that the primary key is composed of all rows. That means, that each triple is unique, thus it is not possible to apply one value twice to the same resource on the same property.

The RDFEditor requires the data and the metadata to be separated into two tables (which may reside in different databases and even on different MySQL servers).

2. Editor Configuration

In the RDFEditor directory is a file called `editor.cfg.php` which stores the global configuration array. Currently, this contains two values: `adminpasswordmd5` and `datamodels`. The `adminpasswordmd5` value is the MD5 encoded password of the "admin" user. This password can be specified as a plain string or by using the PHP `md5()` function.

The `datamodels` value is an associative array where each key is the name of a data model and the value is yet another associative array holding the configuration of that data model. A data model is the logical union of an RDF data source, metadata over that data and some data model options. The RDFEditor can handle multiple (independent) data models. The name of a data model can be any string except `__rdfeum__` which is reserved for the data model storing the user management data.

The keys in the data model configuration array recognized by the RDFEditor are `data`, `metadata` and `options`. The value of `options` is described in section 2.1.

The value of `data` contains information about the data to edit. It is another associative array which must contain the key `type`. Recognized values for `type` are `mysql` and `join`. If `type` is `mysql`, an associative array behind the key `storage` is required in the `data` value array. The `storage` value must have the three keys `link`, `db` and `table`. The last two identifies the MySQL database and table name where the RDF triples are stored. The `link` value is the name of a key in the data model's configuration array. The value of that key is (surprisingly) yet another associative array holding values for the keys `host`, `user` and `pass` which is the information needed to establish a connection to a MySQL server.

If the value behind the `type` key of the `data` value is `join`, an array behind the key `rdfdc` (RDF data collection) is required. The elements of this array specify the single RDF data sources which should be used to compose the joined RDF data. They are of the same structure as the value begin the `data` key of the top-level array. That means that a RDF data join may consist of `mysql` sources or nested `join`s. The keys to the elements are colon separated lists of namespace prefixes. Resources within these namespaces will be written into the associated RDF data sources. If no keys exist (numeric array), the resulting RDF data join will be read-only.

The value of `metadata` contains information about the metadata to use. Its structure is the same as the structure of the `data` value.

Appendix A on page 12 shows a sample configuration file.

2.1. Data Model Options

The value of `options` in the data model's configuration array is an associative array. Recognized keys and the meaning of their values are explained in the following:

defaultNS The default namespace prefix to use when new resources are created. Default: `ns`

hideNSs If true the RDFEditor hides the namespace prefixes when showing URIrefs. Default: `false`

absNS During metadata design some abstract classes have to be declared. Normally the RDFEditor will offer to create instances of that classes which is normally not intended. Therefore all abstract classes should be declared in a seperate namespace which prefix is specified here. Default: *<defaultNS>*`_abs`

hideAbstract If true neither classes nor properties declared in the abstract namespace `absNS` will be shown, thus neither instances of that classes nor values for that properties can be created. This is exactly what is intended by the "abstract" concept, thus the default for this value is: `true`

showNames If true the RDFEditor will show the "name" of nameable resources instead of their URIref. Nameable resources are those which are instances of the `nameClass` (see below). The "name" of these resources is the (first) value on the property which name is specified by `nameProp` (see below). Default: `false`

hideUnnamed This option has only an effect if `showNames` has been set to true. If true those resources which are not instances of the `nameClass` will not be shown in resource lists. Default: `true`

nameClass Name of the class of nameable resources. Default: *<absNS>*`:NameableObject`

nameProp Name of the name property. Default: *<absNS>*`:name`

showHier The RDFEditor can show either a simple resource list which just lists all resources or a hierarchical resource list which shows parents and children of a particular resource. In order to achieve this all the resources which are part of the hierarchy have to be instances of a class which name is specified by `hierClass`. A resource `A` is assumed to be a parent of resource `B` if either `A` has a property `hierPropC` with value `B` or `B` has a property `hierPropP` with value `A`. `A` is assumed to be a child of `B` if either `A` has a property `hierPropP` with value `B` or `B` has a property `hierPropC` with value `A`.

hierClass Name of the class of hierarchical resources.
Default: *<absNS>*`:HierarchicalObject`

hierPropP Name of the "parent" property. Default: *<absNS>*`:parent`

hierPropC Name of the "child" property. Default: *<absNS>*`:child`

hierRootNode URIref of the resource which a hierarchical resource list will display by default.

3. The user interface elements

The RDFEditor user interface consists of several different components which are described in the following.

3.1. Frames

A "frame" of the RDFEditor is an HTML document containing an HTML formular. If multiple browser windows are opened (e.g. the main editor window and a "Select resource" resource list), multiple frames are active which may contain multiple components. The `mainframe` normally shows a create pane, a resource list or a hierarchical resource list (depending on the data model options (section 2.1 on page 3) currently active) and a resource edit pane.

3.2. Create pane

The *create pane* (fig. 1) offers a list of available resource types and a button to create a new resource of the selected type.

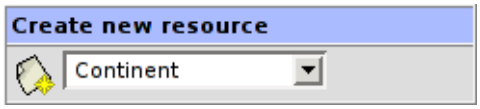

Figure 1: Create pane

3.3. Resource list and hierarchical resource list

The *resource list* simply lists all resources which exist in the currently edited data model. A variant of the resource list is the *hierarchical resource list* (fig. 2) which shows parents and children of a specific resource. The relation "parent" and "child" must be defined in the metadata. The hierarchical resource list shown in the figure shows the hierarchy around the country "Germany". The metadata behind the example defines countries as children of continents and cities and physical societies as children of countries.

Figure 2: Hierarchical resource list

For each resource its type and its URIref or name (depending on the data model options (section 2.1 on page 3) currently active) is displayed. The URIref (or name) is a link which performs an action depending on the context of the resource list. The resource list in the `mainframe` sets the currently edited resource in the resource edit pane of the `mainframe` to the resource which URIref (or name) has been clicked.

The hierarchical resource list additionally shows an arrow ("`-->`") before each resource. By clicking this link you can follow the hierarchy (the resource which parents and children are shown is set to the resource which arrow has been clicked).

On the right side of the hierarchical resource list a "Change resource" button is presented (`[...]`) which asks the user for an URIref and sets the resource lists resource to the one specified by the user.

Since version 1.414213, the hierarchical resource list offers a button to switch to a non-hierarchical resource list ("`[non-hier]`" in the title row). The resource list which is shown after pressing this button offers a button to switch back to the hierarchical resource list.

The list of resources shown by a resource list can be restricted using a filter. By clicking the magnifying glass in the title bar you can open the resource list's filter settings dialog (fig. 3). The RDFEditor distinguishes two different kinds of filters, the type filter and property value filters.

The type filter is set in the upper half of the filter settings dialog. The resource list will display only the resources which type is among the selected types.

The property value filters are configured in the lower half of the filter settings dialog. By clicking the "New property value filter" button a new filter can be created. Existing property filters can be deleted by clicking the "Delete property value filter" button which is shown left to each filter. For each filter three input elements are displayed: The left dropdown shows a list of properties defined in the metadata. The right dropdown offers "regexp" and "equal". If "regexp" is selected in the right dropdown, the value given in the text field is interpreted as a regular expression and the filter reject all resources which do not have a value on the selected property which matches the regular expression. If "equal" is selected the filter rejects all resources which do not have a value on the selected property which exactly matches the string entered in the text field. The comparisons are case-sensitive.

The filter configured in figure 3 matches all resources of type "Country" which have a value on the propery "name" that contains either the string "United" or "land".

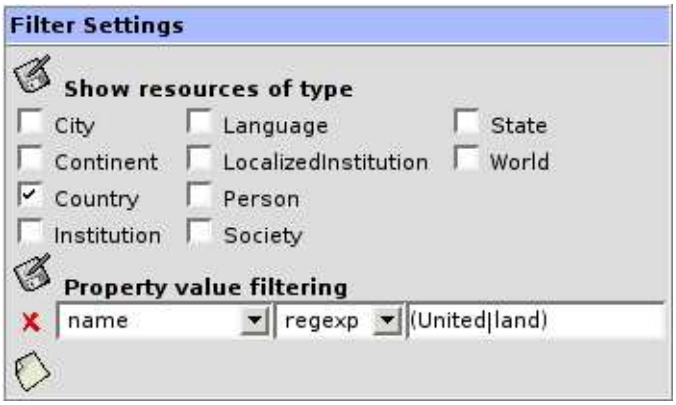

Figure 3: Filter settings dialog

3.4. Resource edit pane

The *resource edit pane* (fig. 4 on page 9) is the facility for editing a resource. The type and the identity of the resource are shown in the title bar (note that the name of the resource is displayed instead of its URIref if the editor option "Show Names" is selected). By clicking

the red cross in the title bar the whole resource can be deleted from the RDF data. This results in the erasure of all triples which subject or object matches exactly the URIref of the shown resource.

The type of a resource can be changed by clicking its type in the title bar. You should not use this feature if you not know what you are doing! The value you enter in the input box will replace the actual `rdf:type` value of the resource. No other properties will be affected thus if the new type does not has a property the old one had all values on that property will remain in the data but uneditable. To avoid this prepare the resource before changing its type by removing all values on properties the new type does not offer.

Left to the resource edit pane's title is a "Change resource" button (`[...]`) which asks the user for an URIref and changes the edited resource to the one specified by the user.

The pane lists all properties which can be applied to the resource and the values on that properties. If a value is a literal value, a text field is displayed. If a value is the URIref of another resource (a reference) the pane may show a sub-resource-edit-pane showing the referenced resource. Since Version 1.4142 the RDFEditor supports the `xsd:boolean` datatype. Values of that type are displayed using two radio buttons "true" and "false". Another special datatype is `rdfe:password`, which is shown as an HTML password input field and which value is stored MD5 encoded only.

Next to the property name the cardinality is displayed. Possible values are "unlimited", "max. 1", "at least 1" and "exactly 1".

If the resource is not editable by the currently logged in user (see 4 on the following page), the resource edit pane shows only the plain data without any edit options. That means, most of the buttons are not shown.

The functionality of each button is explained in the following:

Save changes
All unsaved changes are saved into the data model (this is implicitly performed when any other button is pressed).

Delete value
By clicking on this button the value right to the button is deleted. If the value is a reference to another resource, only the reference is deleted, not the resource. This button is not available if the number of values is equal to the minimum cardinality of the property.

Insert new value
This button adds a new (literal) value on the property. It is not available if the number of values is equal to the maximum cardinality of the property.

Collapse
This button "collapses" the resource edit pane next to it. The value will then show up as a non-editable text field showing the URIref (or name) of the referenced resource and the following three buttons.

Select resource
By clicking this button the referenced resource can be changed. A resource list pops up showing possible alternative resources. By clicking on the resource in the resource list the value is changed to reference the selected resource.

Edit resource
By clicking this button the resource shown in the resource edit pane is changed to the referenced resource (unsaved changes to the resource currently edited will be automatically saved).

Edit resource here
This is the opposite to the "collape" button. After clicking this button the referenced resource will be shown in an embedded resource edit pane making it possible to edit both the currently edited and the referenced resource.

Insert reference to an existing resource
This button adds a new value on the property by selecting an existing resource to reference. A resource list will pop up which shows possible references to select. By using the dropdown right to this button the type of the listed resources can be selected (this is can be changed by editing the filter settings of the resource list). If the data model option "Show Names" is enabled and only nameable resources can be referenced, a text field will be displayed next to the type-select dropdown which value will be used as a property value filter on the name property.

Create new resource
This button is similar to the previous button except that a new resource will be created. The type of the new resource can be selected with the dropdown right to the button. If the data model option "Show Names" is enabled the name of the new resource will be the value of the text field next to the dropdown.

3.5. Container edit pane

The *container edit pane* (fig. 5 on the next page) is a specialized resource edit pane which is designed for editing `rdfs:Containers`. In general, it behaves like a normal property except that the values are numbered. The buttons to the left of the input fields insert or delete values into/from the list (the "Insert" button creates a new value on the current index, all values next to and below the button are shifted one index number). The buttons to the right change the order of the values (these arrows should be quite self-explaining).

4. User Management

Since Version 1.4142 the RDFEditor has been enhanced with an Access Control. It is now possible to allow certain users to edit only allowed data models and—if they are hierarchical—to edit only allowed resources in a data model.

The Administrator (username "admin" and password as specified in the global configuration array in `editor.cfg.php`) can edit the user management data by clicking the "Edit users and permissions" link in the RDFEditor Portal, which is shown after the login. The user management data is stored in the logical data model named `__rdfeum__` (its physical storage is specified in the global configuration array), thus it is RDF data itself.

Each user has to be assigned an unique username (the admin has to take care of uniqueness), a password and a real name. Furthermore each user should get access to some data models. This can be done by adding a `Datamodel` value to the property `allowedDatamodels` or by inserting the user into a group (using the property `isInGroup`) which permissions the

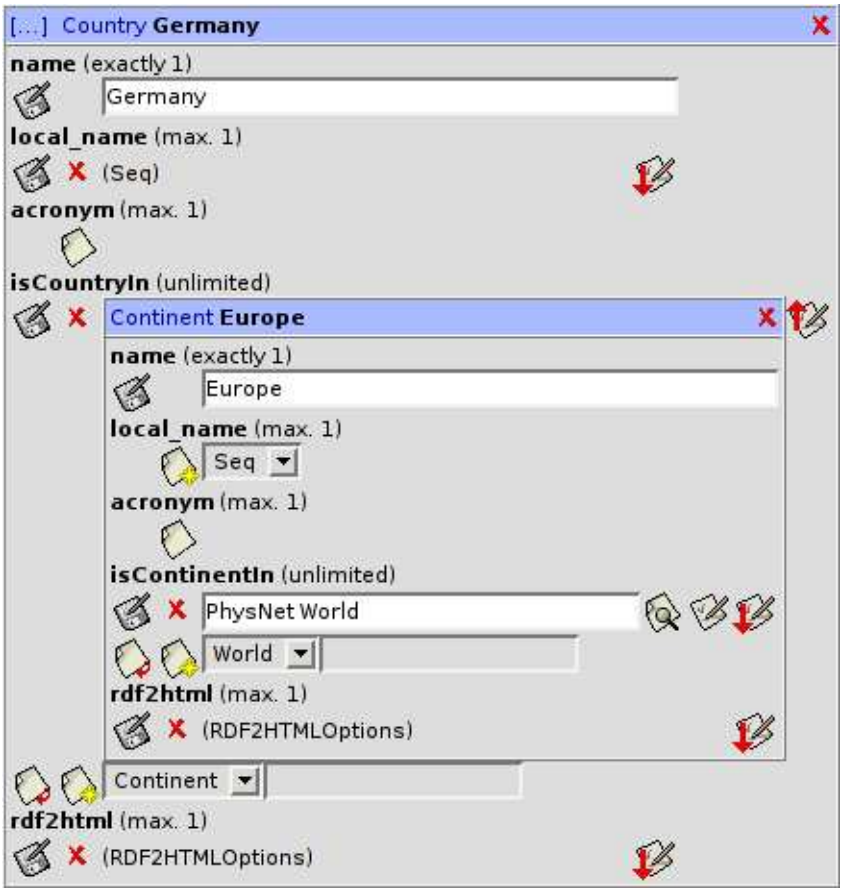

Figure 4: Resource edit pane

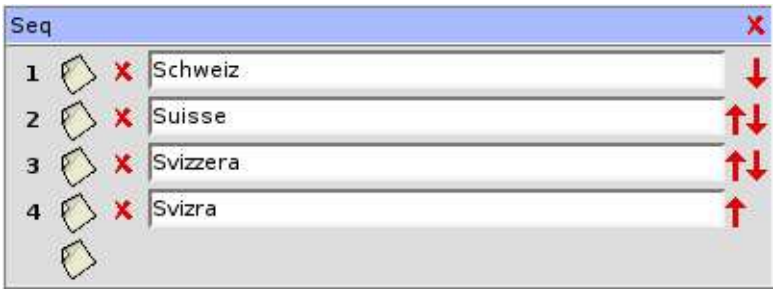

Figure 5: Container edit pane

user should inherit. A group simply consists of a name and some values to the property `allowedDatamodels`. One value on that property allows the access to exactly one data model which name is given in the property `datamodelName` on the `Datamodel` resource. The two other properties `allowedResources` and `restrictedResources` make only sense on data models which have the `showHier` option enabled (see 2.1 on page 3). In that case each resource is tested whether it is editable or read-only. If the URIref of the tested resource is a value on `allowedResources` it is editable, if it is a value on `restrictedResources` it is read-only, if it is not a value on neither of the two properties is is editable if one of its parents is editable or if it does not have any parents. The parents of a resource are the logical parents (specified by the hierarchy) if it is an instance of the `hierClass` or simply all resources which have the URIref of the tested resource as a value on any property.

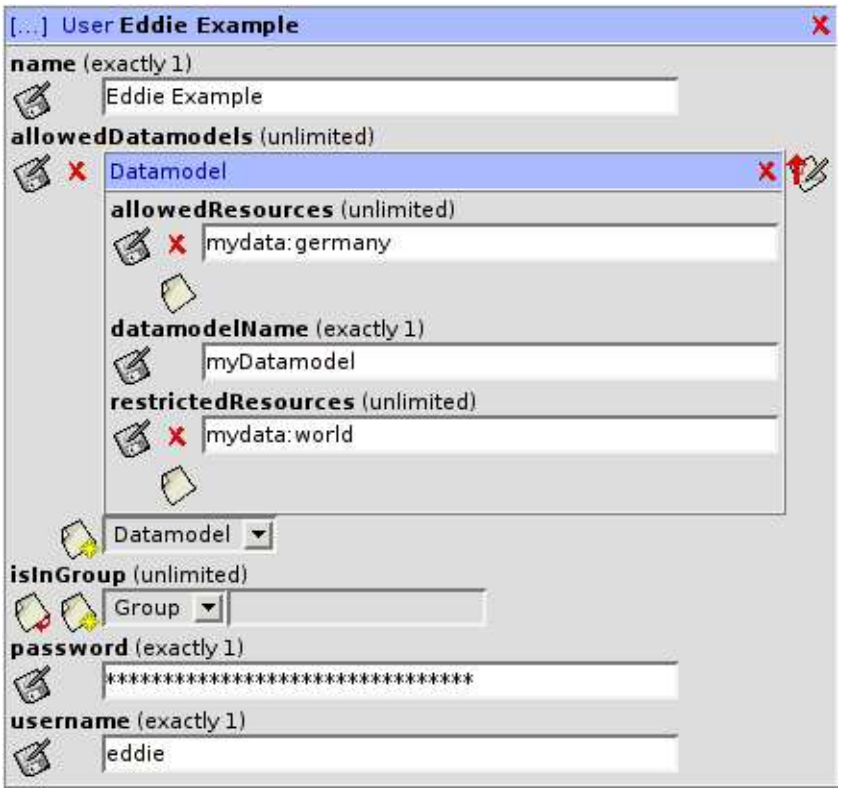

Figure 6: User with restricted write access

Figure 6 shows an example user which has access to the data model "myDatamodel" but is not allowed to edit resources below the resource "mydata:world". In the example data model the resource "mydata:germany" is a child of "mydata:europe" which is a child of "mydata:world". Eddie is allowed to edit "mydata:germany" because it is explicitly meantioned in "allowedResources". Also the whole subtree below "mydata:germany" (which contains for example "mydata:berlin") can be edited by Eddie while "mydata:europe" and other subtrees below "physnet:world" (for example "mydata:asia") can not be edited by Eddie.

References

[1] F. Manola, E. Miller, B. McBride: *RDF Primer*, W3C Working Draft 10 October 2003, `http://www.w3.org/TR/rdf-primer/`

[2] D. Brickley, R. V. Guha: *Resource Description Framework (RDF) Schema Specification*, World Wide Web Consortium, 27 March 2000. `http://www.w3.org/TR/2000/CR-rdf-schema-20000327/`

[3] M. Dean, G. Schreiber, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, L. A. Stein: *OWL Web Ontology Language Reference*, World Wide Web Consortium, 31 March 2003 (work in progress), `http://www.w3.org/TR/owl-ref/`

A. Example: Editor Configuration File

```
<?php
/* RDFEditor configuration file */

$cfg = array(
  "adminpasswordmd5" => "499c208ceafb4fbba162f077060955bd",
  "datamodels" => array(
    "__rdfeum__" => array(
      "mysql_link" => array(
        "host" => "localhost",
        "user" => "physnetuser",
        "pass" => "uiojkl7"
      ),
      "data" => array(
        "type" => "mysql",
        "storage" => array(
          "link" => "mysql_link",
          "db" => "physnet",
          "table" => "rdfeum"
        )
      ),
    ),
    "physnet" => array(
      "mysql_link" => array(
        "host" => "localhost",
        "user" => "physnetuser",
        "pass" => "uiojkl7"
      ),
      "data" => array(
        "type" => "mysql",
        "storage" => array(
          "link" => "mysql_link",
          "db" => "physnet",
          "table" => "physnet_data"
        )
      ),
      "metadata" => array(
        "type" => "mysql",
        "storage" => array(
          "link" => "mysql_link",
          "db" => "physnet",
          "table" => "physnet_metadata"
        )
      ),
      "modules" => array(
        "RDFEditor_PhysNetToolbox.cls.php"
      ),
```

```
47          "session_preset" => array(
48            "frames" => array(
49              "mainframe" => array(
50                "layout" => "CreatePane|createpane||".
51                            "PhysNetToolbox|toolbox|#|".
52                          "HierarchicalResourceList|hierreslist||".
53                            "ResourceEditPane|reseditpane"
54          ))),
55          "options" => array(
56            "defaultNS" => "physnet",
57            "hideNSs" => true,
58            "hideAbstract" => true,
59            "showNames" => true,
60            "hideUnnamed" => true,
61            "nameProp" => "physnet:name",
62            "showHier" => true,
63            "hierRootNode" => "physnet:world"
64          )
65        ),
66        "physnet_copy" => array(
67          "mysql_link" => array(
68            "host" => "localhost",
69            "user" => "physnetuser",
70            "pass" => "uiojkl7"
71          ),
72          "data" => array(
73            "type" => "mysql",
74            "storage" => array(
75              "link" => "mysql_link",
76              "db" => "physnet",
77              "table" => "physnet_data_copy"
78            )
79          ),
80          "metadata" => array(
81            "type" => "mysql",
82            "storage" => array(
83              "link" => "mysql_link",
84              "db" => "physnet",
85              "table" => "physnet_metadata_copy"
86            )
87          ),
88          "modules" => array(
89            "RDFEditor_PhysNetToolbox.cls.php"
90          ),
91          "session_preset" => array(
92            "frames" => array(
93              "mainframe" => array(
94                "layout" => "CreatePane|createpane||".
```

```
95                              "PhysNetToolbox|toolbox|#|".
96                            "HierarchicalResourceList|hierreslist||".
97                              "ResourceEditPane|reseditpane"
98          ))),
99          "options" => array(
100           "defaultNS" => "physnet",
101           "hideNSs" => true,
102           "hideAbstract" => true,
103           "showNames" => true,
104           "hideUnnamed" => true,
105           "nameProp" => "physnet:name",
106           "showHier" => true,
107           "hierRootNode" => "physnet:world"
108         )
109       )
110     )
111   );
112
113   ?>
```